AMENDMENTS TO THE CLAIMS

- At time of the Action: Claims 1-9, 11-15, 17-22, 24-34, 36-43, 45, 51
- Amended Claims: Claims 1, 12, 19-26, 38
- After this Response: Claims 1-9, 11-15, 17-22, 24-34, 36-43, 45, 51

The following listing of claims replaces all prior versions and listings of claims in the application.

1. **(Currently Amended)** A method implemented on a computing device having instructions stored on a computer-readable storage media and executable by a processor, to estimate security requirements needed to execute a managed code for a developer prior to an actual execution of the managed code, comprising:

simulating the execution of all calls from an assembly to another assembly for all execution paths of one or more assemblies in the managed code, wherein [[an]]the assembly comprises one or more files versioned and deployed as a unit, wherein the managed code is a managed shared library or an executable, wherein all managed code is contained within the one or more assemblies, wherein the execution of each assembly is statically simulated without actually running a corresponding managed code to simulate all possible calls and corresponding flow of argument data; [[and]]

finding a set of required permissions for each execution path by one or more simulated stack walks that each include a plurality of the assemblies, wherein each call in each execution path has a corresponding permissions set, wherein each assembly has one or more execution paths representing a different data and a control flow, and wherein the simulated stack walk comprises:

entering an execution path corresponding to a static simulation of execution of the assembly;

entering a public entry point of a method in the assembly;

gathering a permission set for the method in the assembly;

determining whether the method in the assembly calls another method in the assembly or in an another assembly;

gathering a permission set for the ~~called~~ another method called by the method in the assembly; and

creating a union of the gathered permission sets[[.]]; and

deriving the security requirements for execution paths corresponding to the one or more assemblies by using the union of the gathered permission sets across the execution paths corresponding to the one or more assemblies, wherein the union estimates the security requirements that will be triggered against the one or more assemblies during the actual execution of the one or more assemblies and whether a security exception will be triggered during the actual execution.

2. (Original) The method as defined in Claim 1, wherein the execution paths for only one said assembly in managed code are simulated to find the set of required permissions for each said execution path by a union of the permissions for each said execution path.

3. (Original) The method as defined in Claim 1, wherein:

the one or more assemblies in managed code correspond to an application; and

the set of required permissions for each said execution path comprises a union of the permissions for each said execution path.

4. (Original) The method as defined in Claim 1, wherein:

the assemblies in managed code correspond to a shared library; and the set of required permissions for each said execution path comprises one separate permission set per entry point in the shared library.

5. (Original) The method as defined in Claim 1, wherein the set of required permissions for each said execution path comprises a union of the permissions for each said execution path.

6. (Original) The method as defined in Claim 1, wherein one of more of the calls in at least one said execution path is an cross assembly call.

7. (Original) The method as defined in Claim 1, wherein:

the managed code is built to make use of a common language runtime;

each said assembly is packaged as an executable entity or as a data link library entity and

each said assembly includes one or more methods.

8. (Original) The method as defined in Claim 7, wherein the simulation of the execution of each said execution path comprises a simulation of the flow of argument data using intra and extra method data flow analysis for each said method.

9. (Original) The method as defined in Claim 1, wherein when the executable has permissions to execute that are not less than a union of permission sets for each said

execution path, any dynamic execution of the executable will not trigger a security exception.

10. (Previously Cancelled)

11. (Previously Presented) A computer readable storage medium having a tangible component including machine readable instructions for implementing the method as defined in Claim 1.

12. (Currently Amended) In a managed code environment, a method implemented on a computing device having instructions stored on a computer-readable storage media and executable by a processor, comprising:

simulating calling from one assembly to another for which a permission set is required, wherein the simulation comprises one or more simulated stack walks that include two or more of the assemblies, each assembly being managed code in a library, wherein an execution of each assembly is statically simulated without actually running a corresponding managed code to simulate all possible calls and corresponding flow of argument data, and wherein the simulated stack walk comprises:

entering a public entry point of a method in the assembly;

gathering a permission set for the method in the assembly;

determining whether the method in the assembly calls another method in the assembly or in an another assembly;

for each called method:

gathering a permission set for the ~~called~~ another method called by the method in the assembly; and

determining whether the ~~called~~ another method calls a subsequent method in the assembly or in the another assembly; and

creating a union of the gathered permission sets;

repeating the calling for each assembly in the managed code and for all possible execution paths of the managed code;

repeating the entering for each public entry point in the library; [[and]]

finding the union of the permission sets corresponding to each call[[.]]; and

deriving security requirements for execution paths corresponding to the assemblies by using the union of the gathered permission sets across the execution paths corresponding to the one or more assemblies, wherein the union estimates the security requirements that will be triggered against the assemblies during an actual execution of the assemblies and whether a security exception will be triggered during the actual execution.

13. (Previously Presented) The method as defined in Claim 12, wherein the managed code environment comprises:

a managed code portion including:

the assemblies; and

a virtual machine;

a native code portion including:

an execution engine for the virtual machine; and an operating system under the execution engine.

14. (Previously Presented) The method as defined in Claim 12, wherein:

the managed code is built to make use of a common language runtime;

each said assembly is packaged as a data link library entity and

each said assembly includes one or more methods.

15. (Original) The method as defined in Claim 12, wherein when the assemblies corresponding to the application have permissions to execute that are not less than the union of permission sets for each said execution path, any dynamic execution of the assemblies corresponding to the application will not trigger a security exception.

16. (Previously Cancelled)

17. (Original) The method as defined in Claim 12, wherein the managed code environment enforces partial trust security contexts.

18. (Previously Presented) A computer readable storage medium having a tangible component including machine readable instructions for implementing the method as defined in claim 12.

19. **(Currently Amended)** One or more computer~~-readable~~ storage media having a tangible component comprising instructions that, when executed by a processor, perform a simulation of ~~the~~ an execution of every data and control flow for managed code from which an estimate is derived of the minimum security requirements needed to dynamically execute the managed code without triggering a security exception, the instructions comprising: ~~wherein the simulation of the execution comprises, for each data and control flow for the managed code,~~

simulating one or more ~~simulated~~ stack walks for each data and a control flow for the managed code, wherein the managed code corresponds to one or more assemblies, ~~that include~~ wherein the one or more stack walks comprise two or more of the assemblies, wherein the managed code makes use of a common language runtime (CLR) that is loaded upon the first invocation of a routine, and wherein the simulated stack walk comprises:

entering a public entry point of a method in [[the]]an assembly;

gathering a permission set for the method;

determining whether the method calls another method;

for each called method:

gathering a permission set for the called method; and

determining whether the called method calls a subsequent method; and

creating a union of the gathered permission sets[[.]]; and

deriving the security requirements for execution paths corresponding to the one or more assemblies by using the union of the gathered permission sets, wherein the union estimates the security requirements that will be triggered against the one or more assemblies during an actual execution of the one or more assemblies.

20. **(Currently Amended)** The one or more computer~~-readable~~ storage media as defined in Claim 19, wherein:

the managed code, which comprises a plurality of assemblies, is built to make use of a common language runtime;

each said assembly is packaged as an executable entity or as a data link library entity and

each assembly includes one or more methods.

21. **(Currently Amended)** The one or more computer~~-readable~~ storage media as defined in Claim 19, wherein the dynamic execution of the managed code occurs in a managed code environment comprising:

a managed code portion including:

the managed code has one or more assemblies and is a library or an executable; and

a virtual machine;

a native code portion including:

an execution engine for the virtual machine; and

an operating system under the execution engine.

22. **(Currently Amended)** The one or more computer~~-readable~~ storage media as defined in Claim 21, wherein:

the managed code is built to make use of a common language runtime;

each assembly is packaged as an executable entity or as a data link library entity and

each assembly includes one or more methods.

23. (Previously Cancelled)

24. (Currently Amended) The one or more computer~~-readable~~ storage media as defined in Claim 21, wherein:

each call in each simulated stack walk has a corresponding permissions set; and

the derived estimate is a union of the permissions sets.

25. (Currently Amended) The one or more computer~~-readable~~ storage media as defined in Claim 21, wherein the managed code environment enforces partial trust security contexts.

26. (Currently Amended) An apparatus comprising:

means for processing;

means for storing information in memory coupled to the means for processing;

virtual machine means, stored in the memory, in a managed code portion, for operating a plurality of assemblies in managed code, wherein the managed code is a managed shared library or an executable and is in the managed code portion;

execution engine means, in a native code portion, for executing the virtual machine means;

means, in the native code portion, for providing an operating system;

means for making a call in the managed code portion for access by one assembly to another assembly for which a permissions set is required;

means in the managed code portion for gathering the permissions set from each call;

means in the managed code portion for deriving a union of the gathered permissions sets; [[and]]

means in the managed code portion for statically simulating the execution of all possible execution paths for the managed shared library or the executable without actually running a corresponding managed code, to derive therefrom the derived union of the gathered permissions sets wherein the means for simulating the execution performs, for each execution path, one or more simulated stack walks that each include a plurality of assemblies, and wherein the one or more simulated stack walks comprise:

means for entering a public entry point of a method in the assembly;

means for gathering a permission set for the method;

means for determining whether the method calls another method;

for each called method:

means for gathering a permission set for the called method;

means for determining whether the called method calls a subsequent method; and

means for repeating the previous gathering and determining until any gathered permission set is duplicative; and

means for creating a union of the gathered permission sets[[.]]; and means for deriving security requirements for execution paths corresponding to the plurality of assemblies by using the union of the gathered permission sets across the execution paths corresponding to the plurality of assemblies, wherein the union estimates whether a security exception will be triggered during an actual execution of the assemblies.

27. (Previously Presented) The apparatus as defined in Claim 26, further comprising:

means for compiling the assemblies from an intermediate language code and metadata into native code; and

means for loading the native code with a Common Language Runtime loader in the native code portion to load the compiled native code, wherein the execution engine means executes the compiled native code in the native code portion.

28. (Original) The apparatus as defined in Claim 26, wherein the managed code portion further comprises one or more files associated with user code that, when compiled into an intermediate language code and metadata generated by a language compiler, are represented by the assemblies.

29. (Original) The apparatus as defined in Claim 26, wherein the execution engine means in the native code portion further comprises a compiler to compile each said assembly into native code for execution by the native code portion.

30. (Previously Presented) The apparatus as defined in Claim 26, wherein the execution engine means in the native code portion further comprises:

a Just In Time compiler to compile each said assembly into native code; and

a common language runtime loader to load the compiled native code for execution by the native code portion.

31. (Original) The apparatus as defined in Claim 26, further comprising:

means, in the native code portion, for forming a response to the call; and

means for returning the response to the first assembly in the managed code portion.

32. (Original) The apparatus as defined in Claim 26, wherein:

the managed code is built to make use of a common language runtime;

each said assembly is packaged as an executable entity or as a data link library entity; and

each said assembly includes one or more methods.

33. (Original) The apparatus as defined in Claim 32, wherein the simulation of the execution comprises, for each said execution path, a simulation of the flow of argument data using intra and extra data flow analysis for each said method.

34. (Original) The apparatus as defined in Claim 26, wherein when the executable has permissions to execute that are not less than the union of the gathered

permissions sets, any dynamic execution of the executable will not trigger a security exception.

35. (Previously Cancelled)

36. (Previously Presented) The apparatus as defined in Claim 26, wherein each call in each simulated stack walk has a corresponding permissions set.

37. (Original) The apparatus as defined in Claim 26, wherein the managed code portion and the native code portion are in a managed code environment that enforces partial trust security contexts.

38. (Currently Amended) A computing device comprising:

a processor;

a memory coupled to the processor;

a managed code portion stored in the memory including a plurality of assemblies each being managed code in a managed shared library or in an executable;

a native code portion stored in the memory including:

an execution engine ; and

an operating system under the execution engine;

a virtual machine interfaced between the managed code portion and the native code portion and executed by the execution engine;

an application program in the managed code portion comprising logic configured to:

statically simulate the execution of all possible calls from one assembly to another for all possible execution paths of the managed code without actually running a corresponding managed code to simulate all possible calls and corresponding flow of argument data, wherein each assembly call has a corresponding permissions set, wherein the simulation of the execution comprises one or more simulated stack walks that each include a plurality of the assemblies, and wherein the one or more simulated stack walks comprise:

a public entry point of a method in the assembly;

a permission set for the method;

a determination of whether the method calls another method;

for each called method:

a permission set for the called method;

a determination of whether the called method calls a subsequent method; and

a totality of permission sets such that any subsequent permission set is duplicative; and

a union of the permission sets; [[and]]

derive a union of the permissions sets from each assembly call[[.]]; and

derivive security requirements for execution paths corresponding to the plurality of assemblies by using the union of the permission sets across the execution paths corresponding to the plurality of assemblies, wherein the union estimates the security requirements that will be triggered against the one or more assemblies during an actual execution of the assemblies.

39. (Original) The computing device as defined in Claim 38, wherein the managed code portion further comprises one or more files associated with user code that, when compiled into an intermediate language code and metadata generated by a language compiler, are represented by:

the assemblies in the executables; or

the managed shared library.

40. (Previously Presented) The computing device as defined in Claim 38, wherein the execution engine further comprises:

a compiler to compile each assembly into native code; and

a common language runtime loader to load the compiled native code.

41. (Previously Presented) The computing device as defined in Claim 38, wherein:

the managed code is built to make use of a common language runtime;

each assembly is packaged as an executable entity or as a data link library entity; and

each assembly includes one or more methods.

42. (Original) The computing device as defined in Claim 41, wherein the simulation of the execution comprises a simulation of the flow of argument data using intra and extra method data flow analysis for each said method.

43. (Original) The computing device as defined in Claim 38, wherein when the executable has permissions to execute that are not less than the union of the permissions sets from each said assembly call, any dynamic execution of the executable will not trigger a security exception.

44. (Previously Cancelled)

45. (Original) The computing device as defined in Claim 38, wherein the managed code portion and the native code portion are in a managed code environment that enforces partial trust security contexts.

46. (Previously Cancelled)

47. (Previously Cancelled)

48. (Previously Cancelled)

49. (Previously Cancelled)

50. (Previously Cancelled)

51. (Previously Presented) The method of claim 12, wherein the union of the permission sets separately identifies a permission set for each public entry point of the library.